1.Create Restaurant class where there will be 3 thread names: customer, cook and waiter. Demonstrate inter thread communication. [Hint:Create three synchronized methods name: 1.foodOrder(String orderName):will receive the order. 2.cookedOrder(String orderName):will print chef has received the order,The cooking is under process.

Solution:

class Restaurant {

boolean isOrderPlaced = false;

boolean isOrderCooked = false;

synchronized void foodOrder(String orderName) {

if (isOrderPlaced) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.out.println("Customer placed order for: " + orderName);

isOrderPlaced = true;

notify();

}

synchronized void cookedOrder(String orderName) {

if (!isOrderPlaced || isOrderCooked) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.out.println("Cook has received the order. The cooking is under process.");

isOrderCooked = true;

notify();

}

synchronized void servedOrder(String orderName) {

if (!isOrderCooked) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.out.println("Waiter served the order: " + orderName);

isOrderPlaced = false;

isOrderCooked = false;

notify();

}

}

class Customer implements Runnable {

Restaurant restaurant;

Customer(Restaurant restaurant) {

this.restaurant = restaurant;

new Thread(this, "Customer").start();

}

public void run() {

while (true) {

restaurant.foodOrder("Pizza");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

class Cook implements Runnable {

Restaurant restaurant;

Cook(Restaurant restaurant) {

this.restaurant = restaurant;

new Thread(this, "Cook").start();

}

public void run() {

while (true) {

restaurant.cookedOrder("Pizza");

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

class Waiter implements Runnable {

Restaurant restaurant;

Waiter(Restaurant restaurant) {

this.restaurant = restaurant;

new Thread(this, "Waiter").start();

}

public void run() {

while (true) {

restaurant.servedOrder("Pizza");

try {

Thread.sleep(3000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

public class RestaurantDemo {

public static void main(String[] args) {

Restaurant restaurant = new Restaurant();

new Customer(restaurant);

new Cook(restaurant);

new Waiter(restaurant);

}

}

Output:

Customer placed order for: Pizza

Cook has received the order. The cooking is under process.

Waiter served the order: Pizza

Customer placed order for: Pizza

Cook has received the order. The cooking is under process.

Waiter served the order: Pizza

Customer placed order for: Pizza

Cook has received the order. The cooking is under process.

Waiter served the order: Pizza

2. recivedOrder(String orderName):will print the customer received the order.Create one order and call 3 threads named “customer” which will call recivedOrder() method ,”Waiter” which will call foodOrder() method and cook which will call cookedOrder() method and start all the thread.

Solution:

class Restaurant {

boolean isOrderPlaced = false;

boolean isOrderCooked = false;

synchronized void foodOrder(String orderName) {

if (isOrderPlaced) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.out.println("Waiter received order for: " + orderName);

isOrderPlaced = true;

notify();

}

synchronized void cookedOrder(String orderName) {

if (!isOrderPlaced || isOrderCooked) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.out.println("Cook has received the order. The cooking is under process.");

isOrderCooked = true;

notify();

}

synchronized void servedOrder(String orderName) {

if (!isOrderCooked) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.out.println("Waiter served the order: " + orderName);

isOrderPlaced = false;

isOrderCooked = false;

notify();

}

synchronized void receivedOrder(String orderName) {

if (!isOrderPlaced || !isOrderCooked) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.out.println("Customer received the order: " + orderName);

notify();

}

}

class Customer implements Runnable {

Restaurant restaurant;

Customer(Restaurant restaurant) {

this.restaurant = restaurant;

new Thread(this, "Customer").start();

}

public void run() {

while (true) {

restaurant.receivedOrder("Pizza");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

class Cook implements Runnable {

Restaurant restaurant;

Cook(Restaurant restaurant) {

this.restaurant = restaurant;

new Thread(this, "Cook").start();

}

public void run() {

while (true) {

restaurant.cookedOrder("Pizza");

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

class Waiter implements Runnable {

Restaurant restaurant;

Waiter(Restaurant restaurant) {

this.restaurant = restaurant;

new Thread(this, "Waiter").start();

}

public void run() {

while (true) {

restaurant.foodOrder("Pizza");

try {

Thread.sleep(3000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

public class RestaurantDemo {

public static void main(String[] args) {

Restaurant restaurant = new Restaurant();

new Customer(restaurant);

new Cook(restaurant);

new Waiter(restaurant);

}

}

Output:

Waiter received order for: Pizza

Cook has received the order. The cooking is under process.

Customer received the order: Pizza